Approximate BDD Minimization by Weighted A*

Rüdiger Ebendt
German Aerospace Center
Institute of Transportation Systems
12489 Berlin, Germany
Email: ruediger.ebendt@dlr.de

Rolf Drechsler
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
Email: drechsle@informatik.uni-bremen.de

***Abstract*— Reduced ordered *Binary Decision Diagrams* (BDDs) are a data structure for efficient representation and manipulation of Boolean functions. They are frequently used in logic synthesis. The size of BDDs depends on a chosen variable ordering, i.e. the size may vary from linear to exponential, and the existence of a polynomial algorithm to approximate the optimal variable ordering of BDDs implies $P = NP$.**

In this paper, a new approximate BDD minimization algorithm is presented which is based on weighted A*. When compared to the best known previous method, large gains in run time are observed whereas the degradation of solution quality is considerably smaller than for the previous method. The improved behavior now allows for a wider range of time/quality tradeoffs.

Experimental results demonstrate the efficiency of the new approach.

I. Introduction

Reduced ordered *Binary Decision Diagrams* (BDDs) were introduced in [1] and are well known from logic synthesis.

In the past, numerous research papers have addressed BDD-based approaches for the automated design or logic optimization of FPGAs or other multiplexor-based design styles (e.g., see [2]). When mapping BDDs to circuits, a reduction in the number of BDD nodes directly transfers to a smaller chip area. Besides aiming at low area cost, more recent approaches account for other criteria as well (e.g., see [3]). Still all these techniques must not ignore area cost while targeting their particular optimization objectives and therefore use combined criteria. Consequently, the problem of (exact or approximate) BDD node minimization is still crucial for all recent developments.

BDD minimization is not an easy task. It is well known that the size of BDDs is often very sensitive to a chosen variable ordering. In [1] an example has been given where the BDD size varies from linear to exponential dependent on the ordering of the variables. In fact it has been shown that it is NP-complete to decide whether the number of nodes of a given BDD can be improved by variable reordering, and that the existence of a polynomial algorithm to approximate the optimal variable ordering of BDDs implies $P = NP$ [4].

For this, in the past many heuristic approaches have been proposed that are based on structural information or on dynamic reordering of BDDs [5]. But all these methods cannot guarantee an optimal result. In [6] an instructive example has been given where a BDD minimized by sifting has twice the size of an optimal BDD. For the aforementioned applications, this is a significant drawback.

For this reason, exact algorithms have been suggested. The fastest method [7] uses the A* algorithm [8]. However, the run time of the method is still very high and further speed-ups are strongly desirable.

Approximate approaches guarantee a solution whose cost does not exceed the optimal cost by a factor greater than $1 + \epsilon$ where $\epsilon > 0$ is a small number. These methods aim at being faster than their exact counterparts. However, with the nonapproximability result of [4], the run time of an approximate method to improve the variable ordering is expected to be still much higher than that of heuristics.

In [9], a first practical algorithm has been presented that is based on A^*_ϵ, an approximate modification of the A* algorithm. In this paper, weighted A* (WA*) [10] is used for approximate BDD minimization. Like the previous method in [9], the new method provides a parameter ϵ which can be utilized for a time/quality tradeoff. In contrast to the previous method in [9], the tradeoff curve stays monotonically non-decreasing for a wider range of this parameter. Moreover, much higher speed-ups can be achieved while the resulting degradation of quality is smaller than for the previous method. A technical result is given which explains the improved behavior. Experimental results clearly demonstrate the efficiency of the proposed method.

II. Background

A. A* and Weighted A* Search

In this section, the framework of A* and weighted A* search is briefly reviewed.

A prominent goal-directed best-first search algorithm is the well-known A* algorithm [8]. Best-first search is a more general framework of algorithms. The search graph is explored by the use of a list OPEN containing the "open" frontier nodes that have been generated but not yet expanded. A second list CLOSED stores the "closed" inner, expanded nodes. A cost function maps every node to its cost value. A best-first search always expands a most promising open node of minimum cost. Expanding a node means to generate all its child nodes. They are inserted into OPEN, preserving an order based upon the cost values of the nodes (i.e., OPEN functions as a priority queue). The expanded node is inserted into CLOSED. At start, OPEN contains only the initial node and the search stops when a goal node is chosen for expansion. The vertices of the search graph represent the *states* of a *problem state space*. E.g. for the sliding-tile puzzle, a state q is represented by an ordered sequence of tiles. The edges of the search graph describe the possible transitions between the states.

Besides A*, other examples for special cases of best-first search are breadth-first search and Dijkstra's single-source shortest path algorithm. The different instances of best-first search differ only in their cost functions. For A*, the cost of a state q is $f(q) = g(q) + h(q)$. Hereby, two components of information are used with every state q: one is $g(q)$, which is the information about the cost of the path already covered. The other is the heuristic function value $h(q)$, an estimate of the least cost of the remaining part of the path to a goal state. The estimate $h(q)$ has to be a lower bound on the cost of an optimal path from q to a goal state. In this case, h is

called *admissible* and A* is called an *admissible algorithm* since the theory guarantees that A* terminates and always finds a minimum cost path [8]. For a goal state t, it must be $h(t) = 0$. The f-value of t equals the cost of the minimum path $p^*(t)$ which is denoted $f^*(t)$ (or C^*, if t is not of interest). Another property of h, the so-called *consistency*, ensures that the sequence of f-values along every path from the initial state to a goal state is monotonic non-decreasing. It can be shown that A* never reopens expanded nodes if h is consistent. This is important for the efficiency of A*. Noteworthy, consistency implies admissibility.

Besides exact A*, several performance-accelerated extensions are known that sacrifice exactness to gain in run time. A method that still guarantees a bounded suboptimality, can be found in [10]: here, the constant inflation of the heuristic function h by a fixed factor $1+\epsilon$ ($\epsilon > 0$) is suggested. That is, the cost function $f^{\uparrow}(q) = g(q) + (1+\epsilon) \cdot h(q)$ is used instead of the original cost function f of A*.

The method is called weighted A* (denoted WA*). Even if h is admissible, that would not always hold for the inflated heuristic. The admissibility condition of A* is relaxed to direct the search quicker into a more promising direction. Given, that h is admissible, it can be shown that WA* is *ϵ-admissible*, i.e. it always finds a solution whose cost does not exceed the optimal cost by more than a factor of $1 + \epsilon$.

If the inflated heuristic is not admissible, it must also be inconsistent. Therefore states to expanded nodes might be reopened and performance can be degraded. In [11], it has been suggested to modify WA* such that expanded nodes are never reopened again. A proof given in [11] shows that the modified method (called NRWA*) still is *ϵ-admissible*.

B. BDDs

BDDs are well known from logic synthesis. They are a graph-based data structure for the representation of multi-output Boolean functions f: $\{0,1\}^n \rightarrow \{0,1\}^m$. A BDD is a directed acyclic graph where a Shannon decomposition

$$f = x_i f_{x_i=1} + \overline{x}_i f_{x_i=0} \quad (1 \leq i \leq n) \tag{1}$$

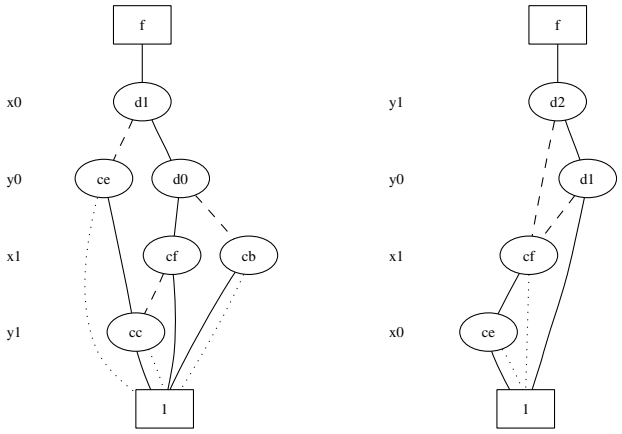
into two cofactors in x_i is carried out with each node. This yields a "then-successor" via a 1-edge and an "else-successor" via a 0-edge, representing the two possible assignments of x_i. An assignment of all input variables corresponds to a path from one of the so-called output nodes to a sink node. The sinks are labeled with the function value for this assignment, i.e. with 0 or 1.

The Boolean variables are from the set $X_n = \{x_1, \ldots, x_n\}$. They are encountered at most once and in the same order (the "variable ordering") on every path from an output node to a sink node. Note that reduced diagrams are considered, derived by removing redundant nodes and merging isomorphic subgraphs. Examples are given in Fig. 1: here, solid (dashed) lines are used for 1-edges (0-edges), and edges to the 0-sink are replaced by dotted edges (so-called *Complement Edges* or CEs, e.g. see [12]) to the 1-sink. For more details see [1].

III. Previous Work

A. Exact BDD minimization by A*

In this paper, approximate BDD minimization is achieved by weighted A*. This approach is based on a previous work [7] which describes exact BDD minimization as a problem of finding a minimum cost path that is solved by A*. To keep the paper self-contained, the basic concept of this work is briefly reviewed in this section.



(a) Initial ordering.

(b) Optimal ordering.

Fig. 1. BDDs for initial and the optimal ordering.

The problem of exact BDD minimization is the problem of finding an optimal variable ordering, i.e. one that leads to a minimum number of BDD nodes. In [7], this problem is expressed as the problem of finding a minimum cost path from the initial state $\emptyset$ to the goal state X_n in the state space 2^{X_n}.

Sets of variables $q \subseteq X_n$ are successively growing from $\emptyset$ to X_n: q is extended at each transition by a variable $x_i \in X_n \setminus q$, i.e. $q \xrightarrow{x_i} q \cup \{x_i\}$. The algorithm starts in the initial state $\emptyset$ and progresses until the goal state X_n is reached. As described before in Section II-A, A* finds a path $p^*(X_n)$ from $\emptyset$ to X_n with minimal cost. The optimal path $p^*(X_n)$ is an optimal sequence of transitions. Consequently, there must exist a permutation σ of the numbers $1, \ldots, n$ (i.e., σ: $\{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ is a bijection) such that the aforementioned minimal cost is the accumulated transition cost for the transitions

$$\emptyset \xrightarrow{x_{\sigma(1)}} \{x_{\sigma(1)}\} \xrightarrow{x_{\sigma(2)}} \{x_{\sigma(1)}, x_{\sigma(2)}\} \xrightarrow{x_{\sigma(3)}} \cdots \xrightarrow{x_{\sigma(n)}} X_n$$

along $p^*(X_n)$. The sequence of variables occurring on this path obviously defines a variable ordering.

The basic idea of the approach is the following: the above ordering annotated along the minimum cost path is intended to be optimal. This means, $f^*(X_n)$ is intended to be the number of nodes in the BDD with the ordering $x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}$. Given, that this already holds, the sequence of variables along $p^*(X_n)$ must be an optimal variable ordering, yielding the minimum BDD size.

To achieve this, an appropriate cost function is chosen. A sequence of variables occurring along the transitions from $\emptyset$ to a non-goal state has the semantics of a *prefix* of a variable ordering. That is, a path of length k defines the positions of the first k variables in a variable ordering. The key idea of [7] is to define the cost function such that the number of nodes in the first k levels of a BDD is taken as the cost of the corresponding path of length k. In this, the method does not perform variable transpositions (as in local search approaches) but incrementally generates the ordering by adding one variable after the other. An example of a run of the A*-based approach of [7] is given in Figure 2. The algorithm is applied to the initial BDD

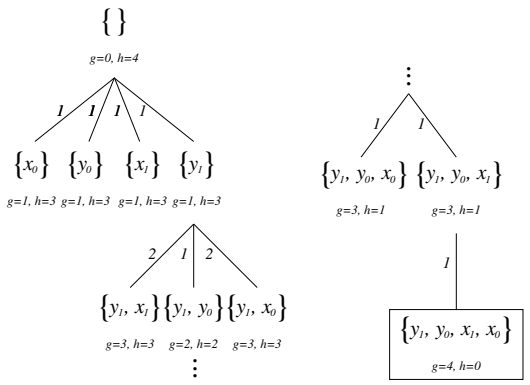


Fig. 2. A* applied to a BDD for a four-input "Achilles heel" function with a bad initial ordering.

in Figure 1(a), which represents the function $f\colon \{0,1\}^4 \to \{0,1\}; (x_0, x_1, y_0, y_1) \mapsto x_0 \cdot x_1 + y_0 \cdot y_1$, an instance of the "Achilles heel" function given in [1]. This function is very sensitive to the ordering. In Figure 2, states are sets of variables which constitute the nodes of the search graph. The g-value and the h-value are annotated at each state. Edges depict state transitions which are always from the top to the bottom. The transition costs (edge costs) are annotated at the edges. At a state q, the heuristic function h counts the number of direct references from the upper nodes in the first $|q|$ BDD levels to the nodes in the lower part of the BDD.

The initial state is the empty set which is expanded to the four successors $\{x_0\}, \{y_0\}, \{x_1\}, \{y_1\}$. The edges leading to them all have costs of 1 because for every successor one root node is established at the first BDD level. Since $g-$ and $h-$values are identical for the first four open nodes, a second-order tie-breaking rule (which, in this case, is motivated by efficiency aspects) selects the state $\{y_1\}$. During the next steps, ties in the value $f = g + h$ are resolved (by the first-order tie-breaking rule) in favor of the state with the lower h-value where possible.

The expansion of state $\{y_1\}$ generates the successors $\{y_1, x_1\}$, $\{y_1, y_0\}$, and $\{y_1, x_0\}$. The order of elements in the set notation gives the path taken from $\emptyset$ to the state (this saves space in the illustration). The successor state $\{y_1, x_1\}$ has a g-value of three. This reflects the total of three nodes in the first two levels of a BDD for the example function given that variable y_1 is situated at the root and variable x_1 resides at the second level. Notice that the structure of the first two (or, in general: k) levels holds regardless of the variable ordering in the part of the BDD below the second (or, in general: kth) level. Due to the partial symmetry of the example function, the BDDs with an ordering $y_1, x_0, \ldots$ have the same g-value of three. In the next step, state $\{y_1, y_0\}$ is expanded since it has the lowest h-value in the set of open states with minimal f-value of four. Again, for reasons of partial symmetry, the BDDs with the orderings $y_1, y_0, x_0, \ldots$ and $y_1, y_0, x_1, \ldots$ have identical g-value of three (and the same h-value of one). From the set of open states with the minimal f-value four, the two states $\{y_1, y_0, x_0\}$ and $\{y_1, y_0, x_1\}$ have the lowest h-value and therefore are selected by the first-order tie-breaking rule. The second-order tie-breaking rule then selects the state $\{y_1, y_0, x_1\}$ for expansion. This results in the optimal ordering y_1, y_0, x_1, x_0 and a BDD with only four nodes (see 1(b)).

B. Approximate BDD minimization by A^*_ϵ

In [9], the framework of exact BDD minimization outlined in Section III-A was used for an approximate method for BDD minimization. The method is called NRA^*_ϵ[1] and is a variant of A^*_ϵ [13] that, like NRWA*, never reopens expanded states. A^*_ϵ equips A* with the capability of terminating earlier with a suboptimal but otherwise perfectly acceptable solution path. This is achieved by adding a second queue FOCAL which maintains a subset of the states on OPEN. This subset is the set of those states whose cost does not deviate from the minimal cost of a state on OPEN by a factor greater than $1 + \epsilon$. More precisely,

$$\text{FOCAL} = \{q \mid f(q) \leq (1+\epsilon) \cdot \min_{r \in \text{OPEN}} f(r)\}$$

The operation of A^*_ϵ is identical to that of A* except that A^*_ϵ selects a state q from FOCAL with minimal value $h_F(q)$. The function h_F is a second heuristic estimating the computational effort required to complete the search. By this the nature of h_F differs significantly from that of h since h estimates the *solution cost* of the remaining path whereas h_F estimates the remaining *time* needed to find this solution.

It can be shown that A^*_ϵ is *ϵ-admissible*, i.e. it always finds a solution whose cost does not exceed the optimal cost by more than a factor of $1 + \epsilon$.

IV. EFFECTS OF RELAXATION

This section gives a new technical result, providing a formal argument in favor of WA* over A^*_ϵ. Later, in Section V, the result is confirmed by the experiments.

Relaxing some of the conditions of A* has certain effects, some of which oppose each other. In [13] it is stated that possibly some states q satisfying the condition $C^* < f(q) \leq (1+\epsilon) \cdot C^*$ are expanded by A^*_ϵ, but not by A*. This effect can exceed the savings of a more focused search and is more likely to be observed for a higher parameter ϵ.

Next, a new result shows that for WA*, the aforementioned negative effect can be much weaker, if often states with equal or similar h-values and/or depth are expanded in a series of consecutive expansions.

Theorem 1: For a snapshot of the progress of WA* with parameter $\epsilon > 0$, consider an optimal path $s, \ldots, q'$ where s is the initial state and q' is the first state that also appears on OPEN. For all states q expanded, either $f(q) \leq C^*$ holds or we have $f(q) > C^*$ *and* $f(q) \leq \mathrm{UB}$ where $\mathrm{UB} = g(q') + (1+\epsilon) \cdot (h(q') - h(q)) + h(q)$. That is, for $h(q)$ within the half-open interval $[0, h(q')($, the upper bound UB ranges from $(1+\epsilon) \cdot C^*$ to C^*, not including $(1+\epsilon) \cdot C^*$ and C^*.

Proof: Because q is expanded before q', $f^\uparrow(q) = f(q) + \epsilon \cdot h(q) \leq f^\uparrow(q')$ ($\diamond$). To derive the stated upper bounds, it now suffices to separate $f(q)$ on the left side of Equation ($\diamond$). The upper bounds range within the stated intervals since the term $h(q')$ can be bounded by $h^*(q')$ because of the admissibility of h, and since an optimal path is considered, we have $g(q') = g^*(q')$ and finally $f^*(q') \leq C^*$. ∎

Similar results hold for the non-reopening variants NRWA* and NRA^*_ϵ. Both has been omitted due to space limitations.

V. EXPERIMENTAL RESULTS

All experimental results have been carried out on a machine with a Xeon processor running at 3.2 GHz, with a main memory of 4 GByte and a run time limit of 3,600 CPU

[1]The original name used in [9] was A^{pprox}.

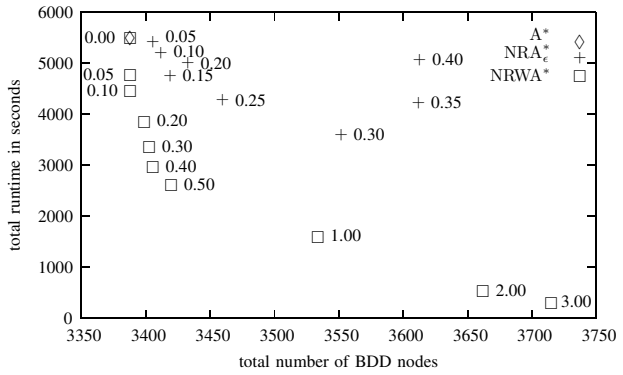


Fig. 3. Trading off run time for solution quality with NRA^*_ϵ and NRWA^*.

seconds. Within this limit, BDDs have been built and minimized for a total of 28 benchmark functions from LGSynth93. The implementation of the new algorithm NRWA^* is based on the implementation of the A^*-based approach of [7]. To put up a testing environment, all algorithms have been integrated into F. Somenzi's well-known CUDD package. By this it is guaranteed that they run in the same system environment.

In a series of experiments, A^* and NRA^*_ϵ have been compared to NRWA^*. The results are depicted in Figure 3. The plot resulting from the experiments with the method NRA^*_ϵ is very similar to a monotonic decreasing hyperbola within the range of 0% up to 30%. The total run time increases again for a degree larger than 30%. Similar results have been observed in [13] where the *Traveling Salesman Problem* (TSP) has been used as a test vehicle for A^*_ϵ: there, as with our application, the number of states expanded often is not a monotonic decreasing function. The reason is that there might be states q satisfying the condition $(1+\epsilon)\cdot C^* \geq f(q) > C^*$ that are expanded by A^*_ϵ, but not by the original A^* algorithm. As Theorem 1 states, this negative effect is much weaker for WA^*, given that often states with equal or similar h-values and/or depth are expanded in a series of consecutive expansions. But exactly this is the case for BDD minimization. The result of Theorem 1 is confirmed by the experiments: in contrast to the behavior of NRA^*_ϵ, the run time of NRWA^* is monotonic decreasing. For NRWA^*, the degradation of solution quality first increases slowly (e.g., for $\epsilon \in [0, 0.5]$) and later ascends more steeply with increasing ϵ. When operating NRWA^* at the turning point of the time/quality tradeoff curve, i.e. with $\epsilon = 0.5$, the total number of BDD nodes was 3420. Compared to the optimal result of 3388 BDD nodes for A^*, the degradation of quality is below one percent. In contrast, at the turning point of the tradeoff curve for NRA^*_ϵ, i.e. for $\epsilon = 0.3$, the highest reduction in run time is observed. The obtained total number of BDD nodes then is 3552. Compared to A^*, the quality is degraded by almost five percent. In this, NRWA^* yields a better quality of the results than NRA^*_ϵ. The improvement is up to almost 19% (e.g., see *tcon* in Table I). When comparing the aforementioned two cradle points in the tradeoff curves of NRA^*_ϵ ($\epsilon = 0.3$) and $\mathrm{NRWA}^*(\epsilon = 0.5)$, NRWA^* has a total run time which is almost 27% smaller than that of NRA^*_ϵ. The gain is up to more than 60% (e.g., see *sct* in Table I).

If higher reductions in quality can be accepted, NRWA^* can be used with higher degrees of relaxation. When comparing NRWA^* with $\epsilon = 3$ to A^*, the loss in quality stays below 10% and a reduction in run time of 94% on average is achieved.

TABLE I
RESULTS OF A^*, NRA^*_ϵ, AND NRWA^*

name	in	out	time $\mathbf{A^*}$	opt	$\mathbf{NRA^*_\epsilon(0.3)}$ time	$\mathbf{NRA^*_\epsilon(0.3)}$ size	$\mathbf{NRWA^*(0.5)}$ time	$\mathbf{NRWA^*(0.5)}$ size
cc	21	20	25.80s	46	15.55s	47	6.77s	46
cm150a	21	1	67.15s	33	16.64s	35	34.60s	35
cm163a	16	5	0.30s	26	0.27s	26	0.20s	27
cmb	16	4	0.09s	28	0.11s	28	0.08s	28
comp	32	3	1371.38s	95	1252.67s	101	745.27s	95
cordic	23	2	0.83s	42	0.67s	44	0.19s	42
cps	24	102	1251.92s	971	328.75s	996	712.85s	987
i1	25	16	9.14s	36	7.30s	36	0.58s	36
lal	26	19	156.30s	67	30.70s	67	48.24s	67
mux	21	1	68.06s	33	16.59s	35	34.37	35
parity	16	1	0.05s	17	0.05s	17	0.04s	17
pcle	19	9	2.34s	42	1.18s	43	0.88s	42
pm1	16	13	0.23s	40	0.20s	41	0.15s	41
s208.1	18	9	2.19s	41	0.95s	44	0.82s	41
s298	17	20	2.93s	74	3.36s	74	1.71s	74
s344	24	26	337.34s	104	359.39s	104	134.76s	104
s349	24	26	334.68s	104	363.92s	104	133.74s	104
s382	24	27	197.58s	119	120.87s	120	74.63s	119
s400	24	27	194.27s	119	120.67s	120	74.30s	119
s444	24	27	173.37s	119	107.38s	120	63.83s	119
s526	24	27	192.96s	113	81.64s	113	49.05s	113
s820	23	24	452.53s	220	341.92s	259	227.53s	224
s832	23	24	455.61s	220	339.90s	259	226.13s	224
sct	19	15	3.14s	48	2.84s	48	0.95s	48
t481	16	1	0.13s	21	0.10s	21	0.09s	21
tcon	17	16	0.26s	25	0.23s	32	0.44s	26
ttt2	24	21	191.89s	107	74.10s	114	54.67s	107
vda	17	39	15.92s	478	12.15s	504	6.59s	479
$\sum$			5508.39	3388	3600.1s	3552	2633.46s	3420

VI. CONCLUSION

A new algorithm for the approximation of the optimal variable ordering of BDDs has been presented. It is based on weighted A^*. We also provide a new technical result which gives a formal argument in favor of the new approach over the best known previous method.

Experimental results are reported that demonstrate the efficiency of the presented approach. A comparison to the best known approximate minimization algorithm shows that the average run time is reduced by almost 27%. At the same time, the resulting loss in quality is up to 19% smaller than with the previous method (on average, the results are now less than one percent away from the optimum). Compared to the best known exact method, an average gain in run time of 94% has been observed.

REFERENCES

[1] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, 1986.

[2] C. Yang and M. Ciesielski, "BDS: a BDD-based logic optimization system," *IEEE Trans. on CAD*, vol. 21, no. 7, pp. 866–876, 2002.

[3] R. Shelar and S. Sapatnekar, "BDD decomposition for delay oriented pass transistor logic synthesis," *IEEE Trans. on VLSI Systems*, vol. 13, no. 8, pp. 957–970, 2005.

[4] D. Sieling, "Nonapproximability of OBDD minimization," *Information and Computation*, vol. 172, no. 2, pp. 103–138, 2002.

[5] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Int'l Conf. on CAD*, 1993, pp. 42–47.

[6] I. Wegener, "Worst case examples for operations on OBDDs," *Information Processing Letters*, vol. 74, pp. 91–96, 2000.

[7] R. Ebendt, W. Günther, and R. Drechsler, "Combining ordered best-first search with branch and bound for exact BDD minimization," *IEEE Trans. on CAD*, vol. 24, no. 10, pp. 1515–1529, 2005.

[8] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 2, pp. 100–107, 1968.

[9] R. Ebendt and R. Drechsler, "Quasi-exact BDD minimization using relaxed best-first search," in *IEEE Annual Symp. on VLSI*, 2005, pp. 59–64.

[10] I. Pohl, "Heuristic search viewed as path finding in a graph," *Artificial Intelligence*, vol. 1, no. 3, pp. 193–204, 1970.

[11] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Formal analysis," Technical report of the Carnegie Mellon University, 2003.

[12] K. Brace, R. Rudell, and R. Bryant, "Efficient implementation of a BDD package," in *Design Automation Conf.*, 1990, pp. 40–45.

[13] J. Pearl and J. Kim, "Studies in semi-admissible heuristics," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, no. 4, pp. 392–399, 1982.